2009:014

MASTER'S THESIS

Estimation of Angular Positions of a UAV from Inertial Measurements

Antonio Francisco Valenca de Affonseca

Luleå University of Technology

Master Thesis, Continuation Courses
Space Science and Technology
Department of Space Science, Kiruna

2009:014 - ISSN: 1653-0187 - ISRN: LTU-PB-EX--09/014--SE

Erasmus Mundus Programme - SpaceMaster

Diploma Thesis

Estimation of Angular Positions of a UAV from Inertial Measurements

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Control Engineering

Luleå Tekniska Universitet

Author: **Antonio** Francisco Valenca de **Affonseca**
Year: 2008

Acknowledgments

The success of this project would not be possible without the help of Tomáš Haniš who helped with the understanding of his mathematical model of the plane. To Jaroslav Žoha for teaching me some tricks and tips using Eagle. I would like to especially thank Martin Rezac who helped me since the beginning of the project in anything that I needed at anytime.

ABSTRACT

This project consists of the design and development of a navigational unit for a UAV (Unmanned Air Vehicle). The purpose is to design from scratch a device that would use three accelerometers, three gyroscopes, a magnetometer and a GPS to provide an estimation of the angular position of the UAV. The tasks involved selecting appropriate sensors, designing a circuit board, assembling the board, connecting the sensors and testing them. A mathematical model of the plane dynamics was also provided to increase the accuracy of the measurement. One extra task was to translate this model, provided in MatLab, into C code.

INDEX

Table of Contents

1 INTRODUCTION 8
1.1 OBJECTIVE 8
1.2 MOTIVATION 9
2 LITERATURE REVIEW 10
2.1 GPS 10
2.1.1 Differential GPS (DGPS) 10
2.1.2 NAVILOCK NL-504ETTL 11
2.2 ADIS16350 11
2.2.1 Gyroscope 12
2.2.2 Accelerometer 13
2.3 HMR3300 13
2.4 EAGLE Software 14
3 DEVELOPMENT 15
3.1 Hardware 15
3.2 Software 19
3.2.1 Low-Level 20
3.2.2 High-Level 25
3.2.3 Program Schematics 28
4 RESULTS 30
4.1 Hardware 30
4.1.1 EAGLE and PCB 30
4.1.2 LPC2119 31
4.2 Software 31
4.2.1 Low Level 31
4.2.2 High Level 33
4.2.3 Combining Low Level with High Level 34
5 DIFFICULTIES 35
5.1 Difficulties 35
5.1.1 Hardware 35
5.1.2 Software 36
5.1.3 Third Party 36
6 CONCLUSION 37
5 FUTURE CHALLENGES 39
7 BIBLIOGRAPHY 41
8 APPENDICES 43
APPENDIX 8. 1 - Declination Angles 43
APPENDIX 8. 2 - NMEA Protocol 47
APPENDIX 8. 3 - MTK Protocol 50
APPENDIX 8. 4 - nelinvyvojP Function example MatLab 51
APPENDIX 8. 5 - nelinvyvojP Function example in C 52
APPENDIX 8. 6 - Substitute1 Function example MatLab 53
APPENDIX 8. 7 - Substitute1 Function example in C 54
APPENDIX 8. 8 - ModelLetA Function example MatLab 54
APPENDIX 8. 9 - ModelLetA Function example in C 55

TABLE OF FIGURES

Figure 1 - Rotational Axis 8
Figure 2 - Board Schematics 16
Figure 3 - Board Layout 17
Figure 4 - Final Board 18
Figure 5 - CPOL and CPHA configuration 21
Figure 6 - SPI Data Sequence 22
Figure 7 - Velocity Vector and Heading 23
Figure 8 - Declination Map for 2000 24
Figure 9 - Maximum rounding error 27
Figure 10 - Program Schematics 30
Figure 11 - HMR3300 Block Diagram 33

TABLE OF TABLES

Table 1 - Example of GGA Data Format (Europe, NaviLock) 11
Table 2 - GYROSCOPE SENSITIVITY (ADIS16350) 12
Table 3 - HMR3300 accuracy (HMR3300) 14

LIST OF ACRONYMS

GPS	Global Positioning System
3D	Three dimensions
INS	Inertial Navigation System
NU	Navigation Unit
PCB	Printed Circuit Boar
DGPS	Differential GPS
MEMS	Microelectromechanical systems
FIR	Finite Impulse Response
UART	universal asynchronous receiver/transmitter
TTL	Transistor–Transistor Logic
GGA	Global Positioning System Fixed Data
NMEA	National Marine Electronics Association
PCB	Printed Circuit Board
ECAD	Electronic Computer-Aided Design
IC	Integrated Circuit
PLL	Phase Lock Loop
MOSI	Master out Slave in
MISO	Master in Slave out
CAN	Controller Area Network
TD1	CAN1 transmitter output
RD1	CAN1 receiver input
SPI	Serial Peripheral Interface Bus
SSEL	Slave Select for SPI
CS	Chip select for SPI
SCK	Synchronous Clock
MSB	Most Significant Bit
GPIO	General Purpose Input Output

XOR	exclusive OR logic
ASCII	American Standard Code for Information Interchange
IDE	Integrated Development Environment
ISR	Interrupt Service Request
MIPS	Millions of Instructions Per Second

1 INTRODUCTION

This chapter has the objective of giving the reader some overall view of what the project is about, its purpose and it will be accomplished.

1.1 OBJECTIVE

This project is part of a bigger project, which is the automation of an unmanned plane. This plane possesses a gimbaled camera attached to it. The main purpose is to lock the camera at a point on the horizon and, as the plane flies around, the camera should automatically update its position to keep it looking at the same point while the plane is moving. Figure 1 shows the final objective. The Navigational Unit needs to provide the P_b, Q_b and R_b angles so that the camera can apply the corrections in order to keep pointing to the same place.

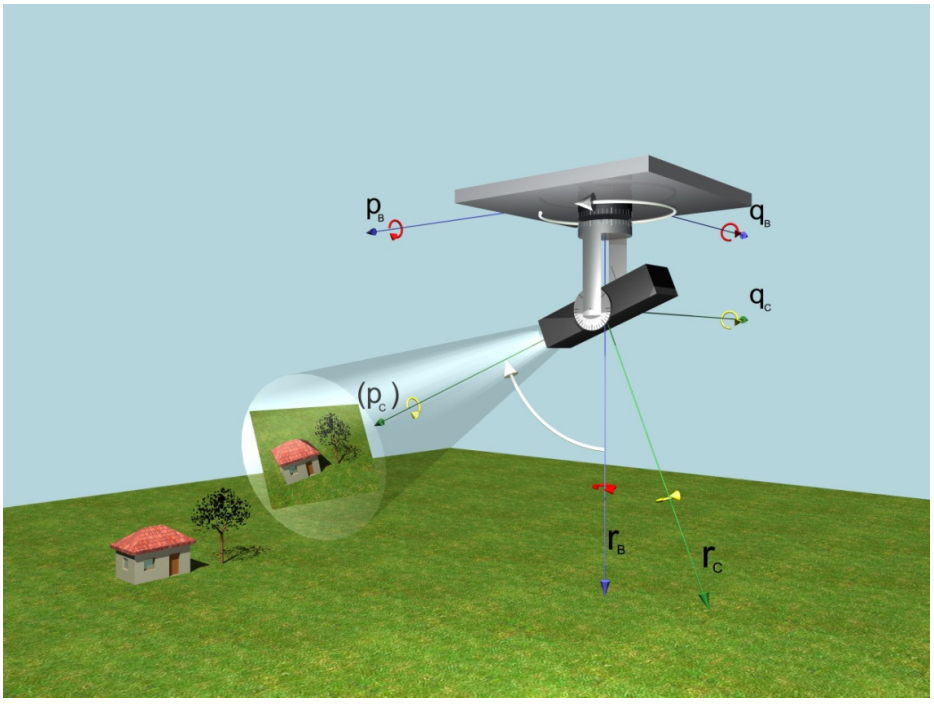


Figure 1 - Rotational Axis

In order for the camera to correctly update its position, it needs to know exactly where it is as a vector in a 3D coordinate plane and where it is supposed to be looking at. This project focuses exclusively at how to precisely tell the plane and the camera where they are located.

The complete project, design and implementation of a GPS aided by an INS and a magnetometer will be the main topic of this project. This should be understood as

- the selection of the sensors for the appropriate needs,
- the design of the PCB,
- mounting of the sensors on the PCB,
- programming of the low-level functions
- programming the high level function
- doing the communication interface with the airplane.

1.2 MOTIVATION

There are innumerous different types and models of NUs constructed with different types of sensors and different data fusion techniques. Each NU is designed specifically for one purpose. The main purpose of the NU is to do the fusion of data from more than one different type of sensor in order to produce one exact output value.

The plane where this NU will be mounted is a small unmanned plane. The main purpose of the plane is to do visualization reckoning of areas where it is possibly dangerous for humans to go. Some places of possible utilization of the plane are: observation of a volcano in eruption or a big fire on a forest. The plane has a simple construction, its maximum speed is of 200 km/h, it is not capable of making aggressive maneuvering changes.

The personal motivations for this project are that it will be possible to put in practice many of the subjects that were thought throughout the SpaceMasters Program. Some of these classes are, but not only, Spacecraft System Design, Space Dynamics, Electronics in Space, Space Systems Modeling and Identification and Estimation Filtering and Fault Detection.

2 LITERATURE REVIEW

Here will be presented the basic knowledge, acquired through the course, for the reader to understand the project and the decisions taken throughout its design.

2.1 GPS

Global Positioning System or GPS is a U.S. Department of Defense constellation of satellites, developed under the NAVSTAR satellite program, that provide radio signals which with the correct equipment will give the position of the GPS antenna. The GPS has as an average of 30 meters of precision for altitude data and 12 meters for North and West plane accuracy and a sample rate of 1 Hz. There are two types of GPS: the most common one has a built in Kalman filter and gives as it output the latitude and longitude among other physical measurements. It can be viewed as a final product to the common consumer. The second type is focused to industry and military or space applications. It is more expensive and much harder to find than a final user GPS. It does not have a built in Kalman filter, instead its output values are pseudorange codes and satellites position coordinates. The pseudorange code includes clock signals from the different satellites locked into the GPS and with the appropriate calculations it is possible to determine your position.

2.1.1 Differential GPS (DGPS)

If one GPS could be placed on a fixed known position, the information from this unit could be used as a reference to nearby non-fixed GPS. This technique is called DGPS or Differential GPS. A station with known position and a GPS transmits a pseudorange code including corrections in real time to other GPSs receivers. These special receivers can then apply corrections to their estimated positions. (Mohinder S. Grewal, January 2007)

2.1.2 NAVILOCK NL-504ETTL

The NL-504ETTL is a complete GPS smart antenna receiver. It has a smart antenna that can track up to 32 satellites at a time with a 1 Hz resolution. It uses a TTL interface which makes it possible to directly connect the GPS with a UART port of the microcontroller. Its speed is fixed at 9600 BPS and a built-in battery makes it possible for the GPS to preserve system data for a rapid satellite acquisition. It has a maximum altitude of 18000 meters and a maximum speed of 515 m/s. The GPS uses the NMEA 0183 ver 3.01 protocol for communication.
Table 1 provides an example of the GPGGA protocol. GPGGA is part of the NMEA protocol. APPENDIX 8. 2 has more information about NMEA protocol.

Table 1 - Example of GGA Data Format (Europe, NaviLock)

$GPGGA,053740.000,2503.6319,N,12136.0099,E,1,08,1.1,63.8,M,15.2,M,,0000*64

Name	**Example**	**Units**	**Description**
Message ID	$GPGGA		GGA protocol header
UTC Time	053740.000		hhmmss.sss
Latitude	2503.6319		ddmm.mmmm
N/S indicator	N		N=north or S=south
Longitude	12136.0099		dddmm.mmmm
E/W Indicator	E		E=east or W=west
Position Fix Indicator	1		See Table 5.1-3
Satellites Used	08		Range 0 to 12
HDOP	1.1		Horizontal Dilution of Precision
MSL Altitude	63.8	meters	
Units	M	meters	
Geoid Separation	15.2	meters	
Units	M	meters	
Age of Diff. Corr.		second	Null fields when DGPS is not used
Diff. Ref. Station ID	0000		
Checksum	*64		
<CR> <LF>			End of message termination

2.2 ADIS16350

The ADIS16350 is a complete solution for an INS. It is built by Analog Devices and it has a complete triple axis gyroscope and triple axis accelerometer. The ADIS16350 is a MEMS sensor, or Microelectromechanical systems. Sensors are mass manufactured at sub-millimeter scales. At such small scale the ratio of surface area to volume

becomes very big, making electrostatic forces significant. Vibration frequencies also scale up making coriolis gyroscopes very effective at MEMS scales. (Mohinder S. Grewal, January 2007)

The ADIS16350 is a highly integrated solution, providing calibrated, digital inertial sensing. An SPI port provides

- X,Y, Z axis angular rates
- X, Y, Z axis linear acceleration
- internal temperature
- power supply
- auxiliary analog input

The inertial sensors are precision aligned across axes, and are calibrated for offset and sensitivity. An embedded controller dynamically compensates for all major influences on the MEMS sensors; thus maintaining highly accurate sensor outputs without further testing, circuitry, or user intervention. (ADIS16350)

2.2.1 Gyroscope

The gyroscope from the ADIS16350 can change its digital dynamic range from ± 75, ± 150 or ± 300 ˚/sec. This means that the maximum rotation that the device can detect is for example 300˚/sec, if that was the selected option. As a trade-off the higher the dynamic range value the smaller the sensitivity will be. Table 2**Error! Reference source not found.** provides the sensitivity values for the determined dynamic range.

Table 2 - GYROSCOPE SENSITIVITY (ADIS16350)

Dynamic Range	Sensitivity
± 300 ˚/sec	0.07326 ˚/s/LSB
± 150 ˚/sec	0.03663 ˚/s/LSB
± 75 ˚/sec	0.01832 ˚/s/LSB

As it can be seen the sensitivity doubles every time the dynamic range is lowered one step. The UAV was designed as a recon vehicle. His normal operating flight mode is very constant and straight forward. It was not designed to make any drastic movements like a missile or a jet fighter, thus it possible to use the dynamic range of ± 75 ˚/sec. This option increases the sensitivity of the sensor by four times when

compared with ± 300 ˚/sec. This will be very important as the sampling frequency will be relatively low, as explained later in this paper.

Each sensor's signal conditioning circuit has an analog bandwidth of approximately 350 Hz. A Bartlett Window FIR filter is available in ADIS16350, which provides an extra noise reduction.

$$H_b(f) = H_a^2(f) \qquad H_a(f) = \frac{\sin(\pi * N * f * t_s)}{N * \sin(\pi * f * t_s)} \qquad 1$$

The frequency response relationship for the filter can be seen in equation 1. N is the number of taps, as a power-of-two step sizes. The use of the filter and its characteristic will be discussed later on this paper. (ADIS16350)

2.2.2 Accelerometer

The accelerometer in the ADIS16350 has a measurement range of ±10g and a resolution of 2.52mg. This resolution cannot be altered and no options can be set. The only available adjustable parameter of the accelerometer is a linear acceleration origin alignment.

2.3 HMR3300

The Honeywell HMR3300 is an electronic compassing solution. These magnetoresistive sensors can be used to provide reliability and accuracy to any solid state compass design. All information sent by the HMR3300 uses ASCII format and the sensor itself can be easily connected to a system using UART or SPI interface. The HMR3300 is a three-axis, tilt compensated compass that uses a two-axis accelerometer for enhanced performance up to a ±60˚ tilt range. (HMR3300)

The HMR3300 when connected through the SPI interface can provide heading information with a resolution of 0.1 degrees. Table 3 shows the complete possibilities of accuracy for the HMR3300 depending of the inclination angle of the sensor.

Table 3 - HMR3300 accuracy (HMR3300)

Accuracy	Tilt	Typical	Units
	Level	1.0	Deg RMS
	0˚ to ±30˚	3.0	Deg RMS
	±30˚ to ±60˚	4.0	Deg RMS

The HMR3300 will send two bytes of data in response to an ASCII "H" or "h." These two bytes represent the integer value equal to 10*Heading. So a value received of 1830 means the heading is actually 183.0˚.

2.4 EAGLE Software

The Eagle software is ECAD software designed by CADSoft, a German company. It is a simple and complete program for designing PCBs. It is divided mainly into two parts. The first one is a schematic editor which is used for drawing the circuit diagrams. The second part is a PCB layout editor. It contains a vast library of commonly used components and ICs and the possibility of making new components for the library. More than 1/3 of the time spent doing this project was in learning how to properly use the EAGLE software. This includes designing the entire PCB, for the integration of the sensors, creating new library pieces and waiting for the board to be manufactured. (CADSoft)

3 DEVELOPMENT

This chapter will show how the development of the project was implemented.

3.1 Hardware

The Philips LPC2119 is an ARM7 type microcontroller. It uses a 10 MHz crystal and through PLL it is able to increase this clock to up to 60 MHz. The LPC2119 has seven input power sources. It is very important that every capacitor, of the corresponding input power source, be placed as close as possible of the microcontroller's input pin. This rule should be applied to every device that has an input power source. If this rule is not followed the device may not work properly or even not work at all. There are three different regulated power sources on the PCB: 1.8V, 3.3V and 5V. The LPC2119 uses 1.8V as the core power supply for the internal circuitry and 3.3V for the pad power supply, or supply voltage for the I/O ports. The 5V is used by the other peripherals on the PCB. (NXP)

Because most of the peripherals on the PCB use 5V logic, like the ADIS16350, the HMR3300, the GPS and the CAN, a voltage translator had to be used. This is necessary because even though the LPC2119 can have an input of 5V, it cannot give an output of 5V, its maximum output is of 3.3V. This voltage level could be misinterpreted by the receiving peripheral. To avoid this possibility, the MAX3379 was used to lift the voltage from 3.3V to 5V. Only the lines leaving the LPC2119 to the peripherals input have the voltage lifted to 5V.

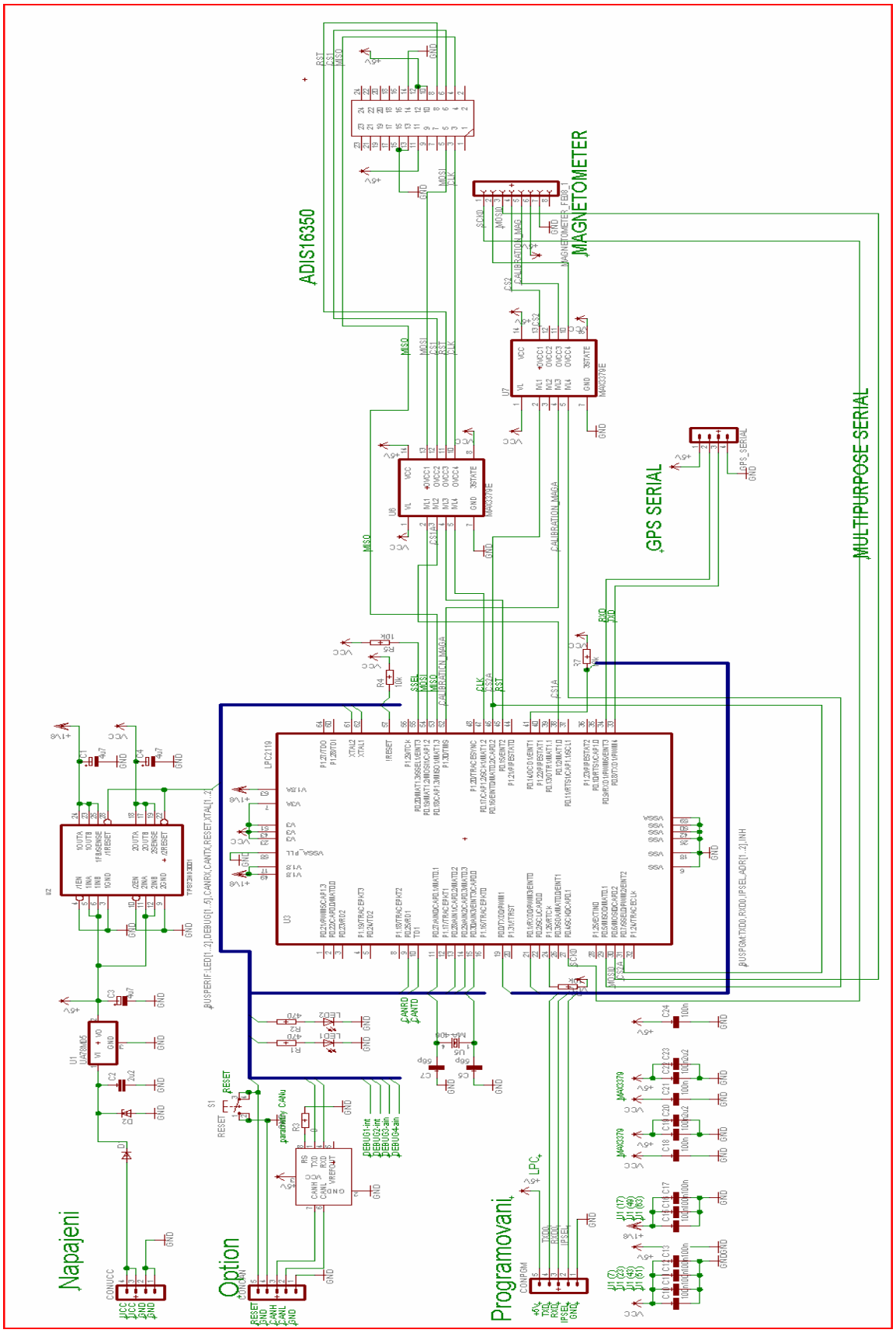


Figure 2 - Board Schematics

The LPC2119 has two SPI ports. SPI1 was exclusively designated for the ADIS16350. This was chosen because the ADIS16350 is in fact six sensors at the same time. They are also the ones that are used the most, the 3 accelerometers and the 3 gyroscopes, requiring a lot of communications between the ADIS16350 and the LPC2119. The ADIS16350 works as a slave SPI, so the LPC2119 is set as the master. For the HMR3300, at first instance, the SPI0 port was selected. The HMR3300 works as a master SPI, which means that the LPC2119 needs to be setup as a slave for the SPI0. (NXP) This also means that the lines between MISO and MOSI in the LPC2119 need to be switched between them. There is the possibility of placing the HMR3300 in the UART0 in case more information is needed from the HMR3300.

Since the GPS works with TTL, it could be directly connected to the UART1 of the LPC2119. The maximum communication speed of the GPS is of 19200 BPS. It will send its GPGGA and GPVTG chain of information every second.

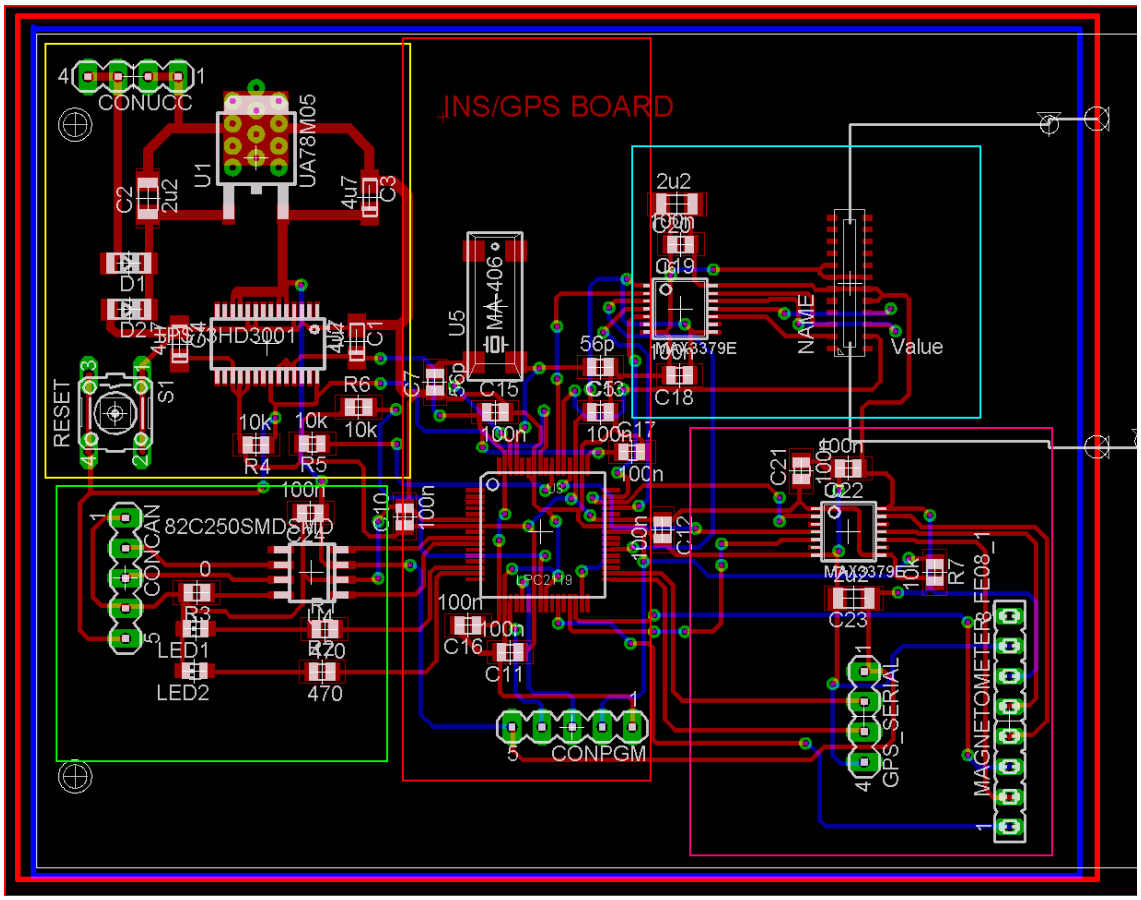


Figure 3 - Board Layout

It is very important to place the GPS, HMR3300 and the ADIS16350 facing the same direction. This will make the software programming a lot easier since they have the

same sign value. For example if the ADIS16350 and GPS are oriented in opposite directions, one would be showing a negative velocity in comparison with the other.

It was selected to use CAN interface for communication with outside peripherals. CAN1 was used but since there is a possibility that the NU will be located far from the outside computer, the RD1 and TD1 had to be connected to PCA82C250 which is a CAN controller interface. It makes the interface between the CAN protocol controller and the physical bus. It will give a differential transmit capability to the bus and differential receive capability to the CAN controller. This will protect the bus against transients in the environment.

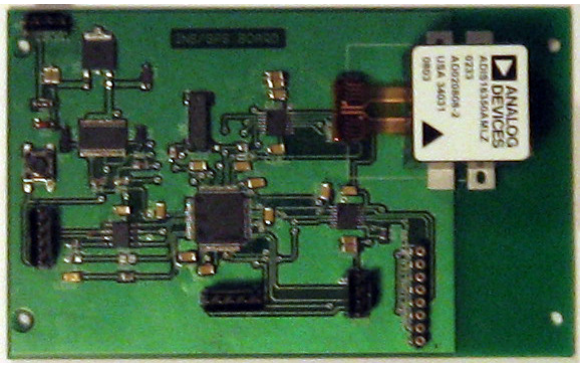


Figure 4 - Final Board

The PCB board layout was finished, after innumerous redesigns, fixes and changes, on 16/04/2008. It was delivered from production on 24/04/2008. With all components in hand the board was finalized and test for use on 26/04/2008. Figure 2 shows the complete board schematics, including the filtering capacitors of the line voltage feeds. All the soldering of the components including the microcontroller was done by hand by the author. A lot of practice was done during the month before in preparation for soldering the final board. The final board can be seen on Figure 4, note that the ADIS16350 can be seen as the white box on the right top corner. Many small details had to be carefully implemented of the board layout. Some of the specifications include; no 90˚ corners on the signal wires and the smallest distances possible from the filtering capacitors and the voltage input lines. The final board was implemented in 5 different zones. This kept the board clean and easy to understand. Also, regulating voltage devices and capacitors are kept far from sensitive parts, like the sensors. Figure 3 shows the board layout and its division by sectors. The components inside the yellow square are dedicated to voltage regulation. The green section concerns the CAN communication. The red box contains the microcontroller, its crystal and filtering capacitors. It also has the connector used to program the microcontroller. The blue box has the electronics concerning the ADIS16350 and the pink box the electronics for the magnetometer. The pink box also includes the GPS port but that is simply a UART port so no extra electronics were necessary.

3.2 Software

The software section will be divided into two subsections. The first part will deal with low level programming, setting up the sensors and putting the data on the correct format. The other will deal with the algorithm for the data fusion and MatLab simulations. In order to program the microcontroller it is needed to place the microcontroller into its boot mode. To do this a special connector is needed that connects pin1 to pin2 of the CONPGM connector located in the lower part of the red box in Figure 3. After that the reset button on the yellow box needs to be pressed.

Half of the program, specifically the low level programming, was written directly on a Linux machine preloaded with a university written software to compile the program and write it and run it through the RAM of the microcontroller. This provided a fast way of testing small pieces of the program in real-time situations, saving time and life cycles of the microcontroller, since the microcontroller has a limited number of writings to the flash memory. The drawback is that the RAM memory of the LPC2119 is very small if compared with the flash memory. The LPC2119 has up to 16Kb of RAM and 256 Kb of flash memory. More about this will be explained at the end of this section after the high level programming is explained.

The second part of the program is defined as the high level programming, or the simulation and modeling of the airplane. This part involves translating the MatLab file of the model and simulation into C code. This will be better explained on High Level section below. This part does not actually need any real data from the sensors in order to simulate the output so a more user friendly program was used to write it than the Linux gcc. The program chosen was DeV-C++, a free IDE for programming C/C++. It runs exclusively on windows. This software provides easy debugging functions and screen visualization of step by step line execution of the code. This provided a very useful and powerful tool for comparing results between the original MatLab model and the C translated code. The main problem when changing the code from the Dev-C++ to the Linux program, in order to test the final software inside the microcontroller, was that the size of the RAM was too small to fit all the code. Public functions like cosine and sine, included in libraries like MATH.h or other, provided a particular strain on memory resources. These functions are very general and need to have a lot of special programming in order to work in every code, making it a very big sized function. This added to the fact that the code uses many 22x22 matrices made it impossible to use the RAM for testing. Another approach had to be considered. It was needed to actually write the program into the

flash memory. For this it was necessary to debug and save the file in HEX format with the Linux software and use windows software to write it into the flash which became very time consuming.

3.2.1 Low-Level

The HMR3300 is the master in the SPI interface. This means that the SSEL0, which is the pin that puts the SPI0 in the LPC2119 as master or slave, needs to be pulled down. This is accomplished by connecting the CS of the HMR3300 together with the SSEL0. So when the CS is lower so will SSEL0 be lowered. This will at the same time set the LPC2119 SPI0 to slave mode and select the HMR3300 ports. Since the HMR3300 is the master, it will select the clock speed automatically. To ask for the heading it is needed to send an "h" character to the HMR3300. This will write the "h" to the register in the HMR3300. The HMR3300 operates with SCK idle low. Data output comes after the falling edge of SCK. Data sampled comes before rising edge of SCK. This is simply done by adjusting the CPOL and CPHA registers of the SPI0 port both to 0. Figure 5 shows the possibilities of configuring the CPHA and CPOL registers.

As soon as CS of the HMR3300 is lowered, the HMR3300 will send an "s" character to the LPC2119. In response the LPC2119 must send an "h" character back. After this the HMR3300 will start sending the heading data at regular intervals.

Contrary to the HMR3300 the ADIS16350 is a slave SPI sensor. It uses the LPC2119 as a master.

Because of this a number of setups need to be done in order for the ADIS16350 to work properly. First the ADIS16350 sets both CPOL and CPHA to 1. This means that the first data comes with the first SCK falling edge. Other data driven with SCK falling edge and the data sampled with SCK rising edge.

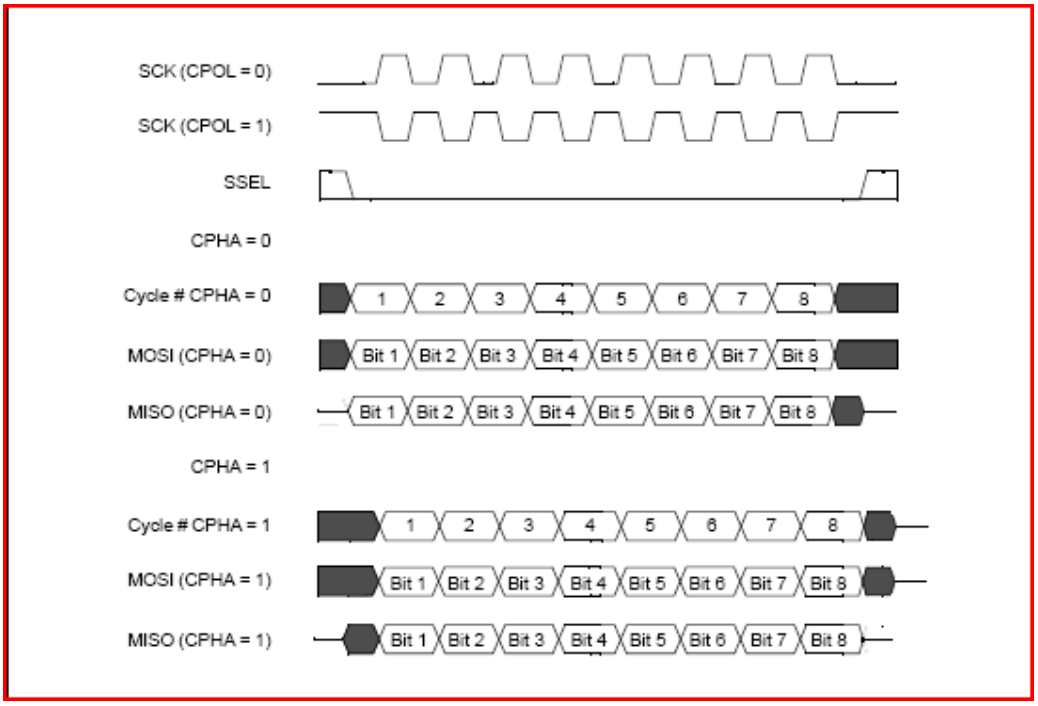

SCK (CPOL = 0)
SCK (CPOL = 1)
SSEL
CPHA = 0
Cycle # CPHA = 0
1 2 3 4 5 6 7 8
MOSI (CPHA = 0)
Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6 Bit 7 Bit 8
MISO (CPHA = 0)
Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6 Bit 7 Bit 8
CPHA = 1
Cycle # CPHA = 1
1 2 3 4 5 6 7 8
MOSI (CPHA = 1)
Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6 Bit 7 Bit 8
MISO (CPHA = 1)
Bit 1 Bit 2 Bit 3 Bit 4 Bit 5 Bit 6 Bit 7 Bit 8

Figure 5 - CPOL and CPHA configuration (NXP)

It was decided to use a SCK of 600 KHz with 8 bits data, no parity bit with one stop bit, with MSB and interrupt enabled. This can be simply done by setting:

S1SPCCR=24;

And

S1SPCR=0xB8;

Besides these configurations it is only necessary to pull-up the CS pin of the sensor with an ordinary GPIO pin from the LPC2119. For more information on the registers and how to calculate the baud rate and other configurations see (NXP).

Both HMR3300 and ADIS16350 use the same function to receive and or send data. It simply stores a value and an address to a registry. After it is sent it uses an interrupt function to receive the data from the prior command sent. Figure 6 shows exactly how the information sent through SPI works. This example is from the ADIS16350 but the same can be applied to the HMR3300 with some minor changes already mentioned.

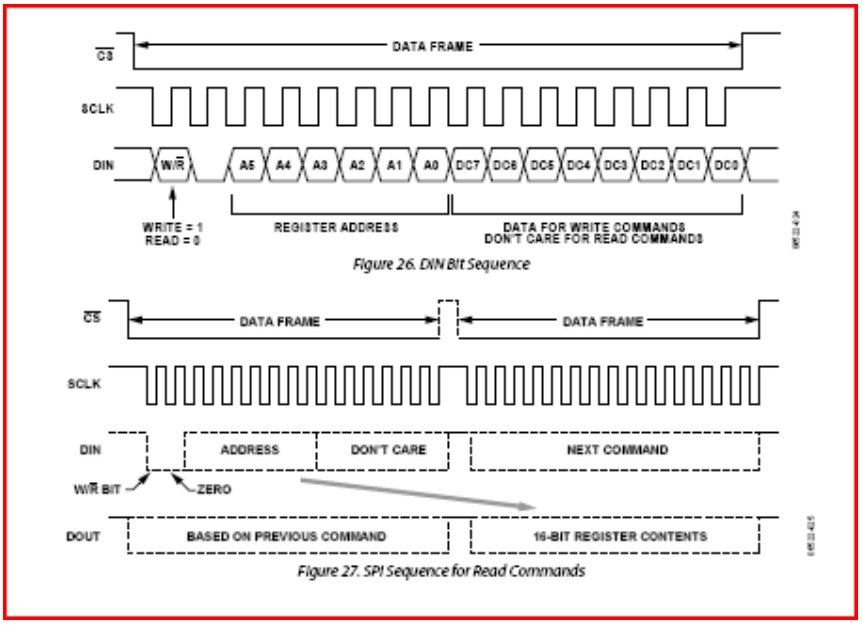


Figure 6 - SPI Data Sequence (ADIS16350)

One timer is used to set a fixed sampling frequency for the sensors. TIMER0 was set to make an interrupt every 0.1 seconds or 10Hz. When TIMER0 interrupts it sets a flag that runs a routine to acquire the data from all the sensors from the ADIS16350. The 10 Hz frequency was selected because it is a reasonable good number of points taking into consideration the heavy matrix multiplication that will go on in such a weak processor.

The GPS uses the NMEA protocol for sending data to the host device as mentioned before. In order to program the GPS it's needed to use the MTK protocol. It has the same exact format as the one shown in Table 1 except that instead of having the header $GPXXX (where XXX correspond to the desired output data) it has $PMTK. All writing commands and acknowledgement commands from the GPS use the header $PMTK. This provides a way to differentiate what is GPS data from what is configuration data. Like the NMEA protocol, MTK protocol also has a checksum at the end of every command. This checksum is a simple consecutive XOR of the first character after the "$" until the last character before "*." In order to calculate this checksum a simple program was developed to automatically calculate the checksum of the command line and add this value to the two characters after "*". Also every command line should end with a <CR><LF> which is nothing more than a 0x0D and 0x0A from the ASCII table. APPENDIX 8. 3 has more information about MTK protocol and commands.

The GPS was then configured to transmit at 19200 BPS which is the maximum baud rate of the GPS. The GPS was set to send only $GPGGA and $GPVTG strings every

second. The GGA format will provide an altitude value and a corresponding time. Equation 2 can be used to derive the Z-axis velocity.

$$\frac{(\mathbf{Altitude_2} - \mathbf{Altitude_1})}{(\mathbf{Time_2} - \mathbf{Time_1})} = \mathbf{Velocity_Z} \qquad 2$$

The GPVTG string will provide heading information with respect to either Magnetic or True North. Both options are available. It will also provide the ground speed with respect to its heading. Figure 7 - Velocity Vector and Heading provides a good visualization for the relationship between the ground speed and the heading angle H or velocity vector over a 2D plane or XY plane.

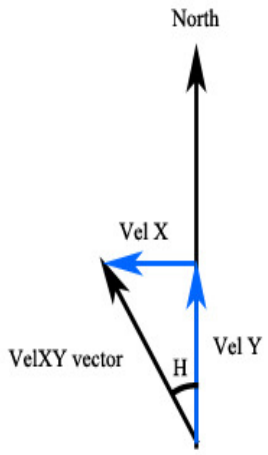


Figure 7 - Velocity Vector and Heading

Using simple mathematics, seen on equations 3 and 4 it is possible to break down the velocity vector VelXY into two orthogonal vectors, VelX and VelY.

$$\mathbf{Vel_x} = \sin(\mathbf{H}) * \mathbf{Vel_{XY}} \qquad 3$$

$$\mathbf{Vel_Y} = \cos(\mathbf{H}) * \mathbf{Vel_{XY}} \qquad 4$$

Initially the HMR3300 was going to be connected through the SPI interface but because of the complexity of the SPI slave mode for the LPC2119 and complex timing

diagrams for the HMR3300 it was decided to use the UART0 interface. With this the same UART functions for receiving values or writing values could be used for acquiring or sending information for the HMR3300, saving a lot of designing time. Also as it was found out that the HMR3300 needs to be setup with a declination angle correction for the place where it will be used. The declination angle is the angle between your local magnetic field, also known as the North direction end of the compass needle, and the True North. Magnetic declination varies from one place to another and from time to time, Figure 8 provides the declination map for the year 2000. If for example the plane was set for Prague but flew to Warsaw it would have an error in its heading of 2 degrees. For a list of world capital cities declination angles see APPENDIX 8. 1 (Timex).

During the last 15 days of the project it was noticed that the Magnetometer was not working properly. It was sending correct data structure but the values were not coinciding with what it should be. After doing some test it was found out that the Magnetometer was not correctly calibrated. Since there was no time to go fix it, the code was changed to use the heading provided by the GPS. But the code from the Magnetometer is ready for when it gets fixed.

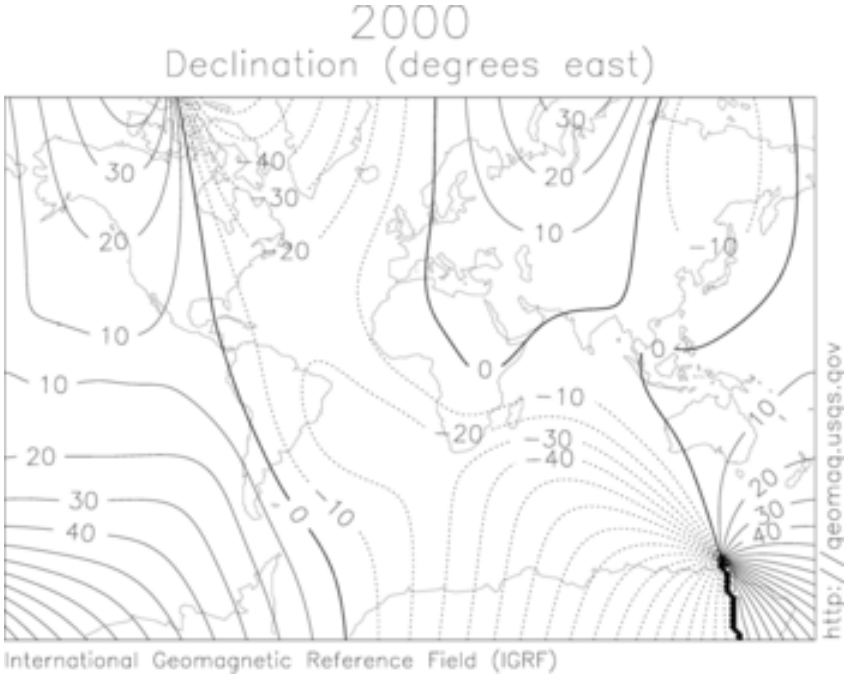


Figure 8 - Declination Map for 2000

3.2.2 High-Level

After all the low level is software is run the following sensor data is available from the ADIS 16350 sensor:

- X acceleration
- Y acceleration
- Z acceleration
- X angular speed
- Y angular speed
- Z angular speed

From the GPS and Magnetometer it is possible to acquire X velocity, Y velocity, Z velocity and heading.

The high-level program consists of the implementation of a model of the plane where the Navigational Unit will be installed. The model was developed by Tomáš Haniš and consists of five files. Each file was translated from MatLab into C code. This procedure took about two weeks. Every file became a function and all main matrices were declared as global. The first one was "substituce1." This file consists mainly of the A matrix from the model. It was the hardest file to translate. The following problems appeared during the translation process:

- Use of extremely big integers that were not supported by the "float" type.

  For example: 6707078289772873775 **/** 9511602413006487552

- Use of extremely complex equations that resulted into the following error message from the gcc "equation is too complex." For example the equation for one element A[3][22] had a total of 15700 characters. This is equivalent to almost 5 full pages using font 11.
- Multiple use of exponential, sine and cosine functions.
- Just this file, A matrix, took 0.089 seconds to be set up using a Core2Duo 2GHz computer. This is almost 0.1 second which is the total time allowed for one entire main loop to occur.

The total size of Matrix A was of 57300 characters which is the equivalent of 28 pages - the matrix required extreme optimization. The first step was to replace the division between two very big integers by one float value. So that 6707078289772873775/9511602413006487552 became 0.705147040271720. This not

only dealt with the size of the integer which was too big to fit in a "float" type but it also reduced the matrix by one less division step. After that many simpler division operations were also substituted by their final value. This was by far the most time consuming part of the code translation. It took around 6 days just to convert Matrix A into C code and optimize it.

Since the matrix A is updated only every cycle loop, the sines, cosines and exponential functions use the same angles. Knowing this it is possible to calculate the sine, cosine and exponential functions only once at the beginning of the function and store the values as constants. This procedure was not only done to sine, cosine and exponential functions but also squared and cubic functions were substituted by a constant and later used to substitute multiple appearances of the same value in the matrix. A cascade technique was used where smaller constants were substituted inside bigger constants and these into bigger constants.

After the optimization was finished the matrix A size was reduced to 11200 character or the equivalent of 8 pages long. More important than the size, the time to setup matrix A was reduced from 0.089 seconds to 0.016 seconds which is 5.56 times faster than the original code. The maximum rounding error found during simulations was on the order of $3.5*10^{-5}$ which is totally acceptable. Figure 9 below provides the maximum round error found during simulations using around 6000 steps. An example comparing the results with MatLab file and the translated C code can be seen in APPENDIX 8. 6 and APPENDIX 8. 7. For this random input the files have perfect match up to four significant digits.

The second file that became the second function is the nelinvyvojP which is simply the update of matrix P, the covariance matrix, using the equation 5.

$$P=P+(A*P+P*A'+Q)/deltaf/10 \qquad 5$$

where deltaf is the sampling time. In APPENDIX 8. 4 and APPENDIX 8. 5 there is the result of the function nelinvyvojP both for MatLab and with its C translated code. As it can be seen the results are exactly the same.

The third file is the NaplnMSC, also this file is the simple Direct Cosine Matrix. The next file is the ModelLetA. This function is the actual model of the plane. The

function contains all constants and inputs to the states. In the APPENDIX 8. 8 and APPENDIX 8. 9 it is possible to compared the output of the ModelLetA function both in MatLab and its translated code in C. One more time the functions are identical up to four decimal places.

The last file is the ExKalman2A, this is the main function and calls all the other functions. The Kalman filter is always implemented in this file. The key trick in this file is whether the loop that is running has new GPS data or not. More will be explained later. One of the most difficult parts in translating the code from MatLab to C was the function that solves the inverse of a Matrix or $\mathbf{A^{-1}}$. This can be seen on equation 6, seen below, or the Kalman gain of the model of the plane.

L = Pd*Cgps'*inv(Cgps*Pd*Cgps'+Rdgps) 6

The solution was to use the Cholesky process. This decomposes a symmetric positive-definite matrix into a Lower Triangular matrix and the transpose of this Lower Triangular Matrix. A positive-definite matrix is defined by having all Eigenvalues of the matrix to be positive. Symmetric is defined by a square matrix that is exactly the same as its transpose, $\mathbf{A = A^T}$.

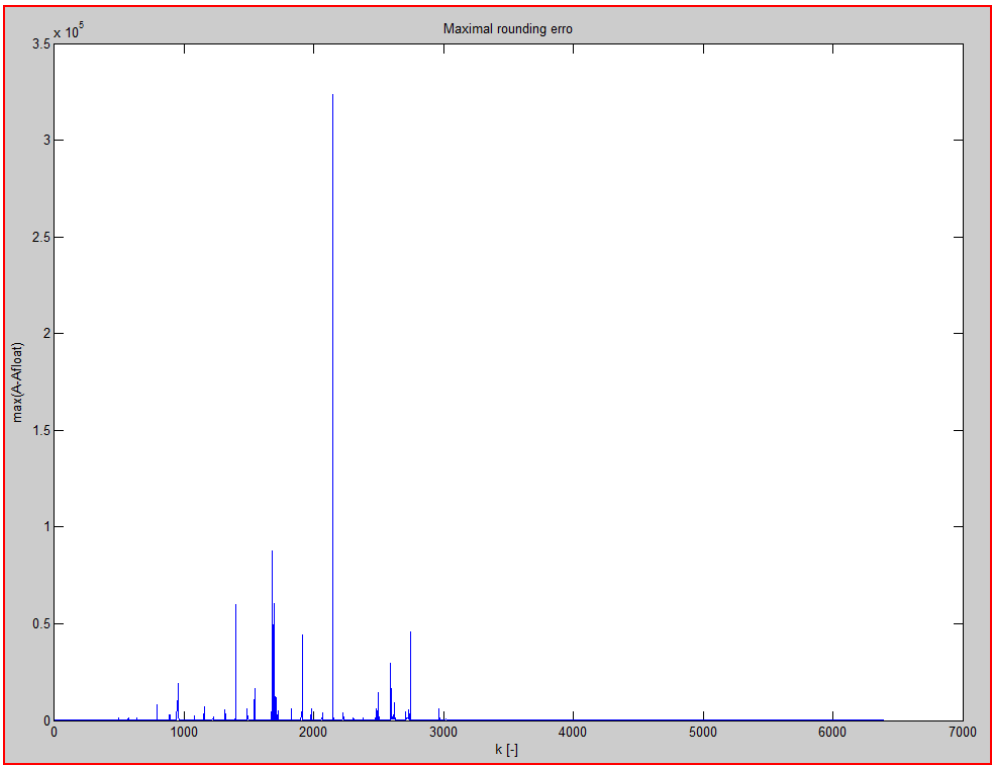


Figure 9 - Maximum rounding error

Equation 6 can be simplified to:

$$\mathbf{L = B * A^{-1}} \qquad 7$$

This could be finally be rewritten into

$$\mathbf{L*A=B} \qquad 8$$

Where A is the matrix to be decomposed, L is the unknown variables and B the solution to the equation. From equation 8 the Cholesky decomposition technique can be directly applied. It states that to solve A*X = B it is possible to first compute the Cholesky decomposition $A = L*L^T$, then solve L*Y = B for Y, and finally solve $L^T*X = Y$ for X. Assuming that matrix A is a symmetric positive-definite matrix.

Two different gains could be calculated, one if there was new GPS information, that would run once every second, and one when there was no GPS information, which would run 9 times per second. A simple "flag" variable was used to determine whether or not new data was available in the UART buffer and select the GPS path; otherwise the non-GPS data would run.

3.2.3 Program Schematics

As explained before the program is mainly divided into a low level section and a high level section. The follow through of the program as it goes from startup would be equivalent to Figure 10. The first six steps, from `Initialize Libraries` to Setup GPS Protocol, only happens once at startup. `Initialize Libraries` contains the main libraries for the program to run, including `math.h`, `stdio.h`, `can.h`, `uart_zen.h` among others. `Declare Global Variables` contains the main variables from the low level programs as well as all major matrices and constants from the high level code. All the functions designed specifically for the NU are declared on the main coding file under `Declared Functions`. `Initialize`

`Peripherals` consists of all LPC2119 initialization. This includes the UART, SPI, CAN, GPIOs, Levels, Baud Rates, interrupts and timers setups.

The main loop start includes the "GPS Flag = 1?" section. It will keep looping without doing anything until it receives the ADIS16350 flag=1. Just the ADIS flag=1 will enable the code to treat the data from the sensor and to run the filter option without the GPS data. This process occurs once every 0.1 seconds. In case the GPS sends data the UART interrupt will be enabled. This will save the data from the GPS and treat it. It will also enable the GPS new data flag and as soon as the ADIS16350 flag is set the path for running the filter with GPS data will be enabled, and at the same time the path without GPS data will be disabled. After both filter processes are complete flags from the GPS and from the ADIS are set to zero. The program enters an idle while loop until one of the flags is activated again.

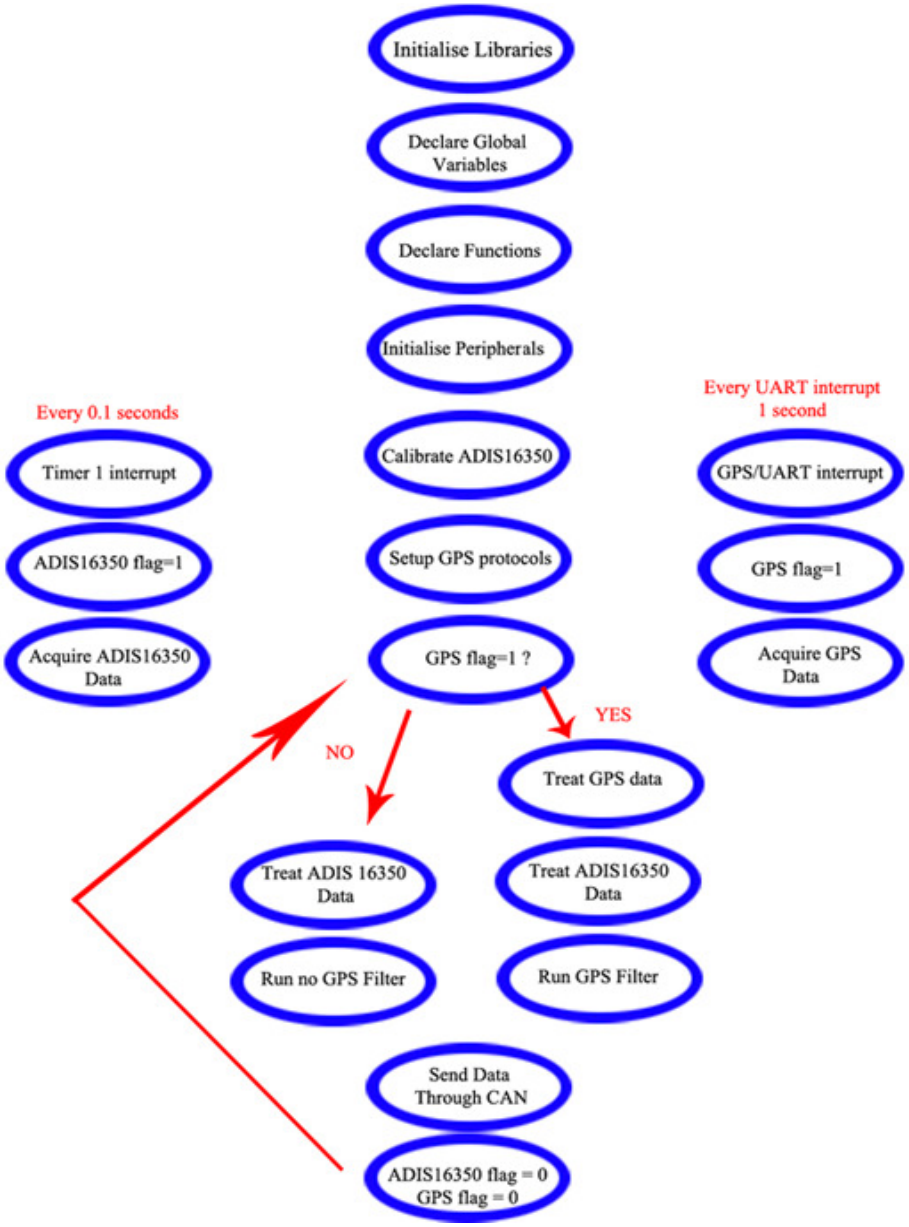


Figure 10 - Program Schematics

4 RESULTS

Brief description of the final results obtained as well as the results of smaller individual topics that contributed to the overall project. .

As this project consisted of many different independent parts many results were obtained. Most trials were successful; others are still on the way to be finalized as the report is being written. As part of the project was to actually acquire some deeper knowledge on a more specific topic related to the master course, the knowledge acquired by the author will also be presented.

4.1 Hardware

4.1.1 EAGLE and PCB

The hardware portion of the project can be said to be the most successfully accomplished part of the project. In terms of the educational gain the author acquired a full understanding of the EAGLE software where before he had absolutely no knowledge of the software about any other technique for building PCBs. The ability to almost master the software is one of the most important accomplishments on the author's viewpoint. Besides building the schematics to a PCB there is also the complexity of designing the board by itself. Another point was the necessity of learning how to design new hardware components since the ADIS16350 is a new sensor and its library was not yet available.

4.1.2 LPC2119

The ARM7 LPC2119 microcontroller is not an easy microcontroller to learn how to use. It has complex ways of dealing with different types of interrupts. For example an interrupt needs to have an address and a priority level over other interrupts. This address will point to a designated place of your code where the ISR, or the code you want to run when interrupt is activated, is placed. After executing you need to clean the interrupt flag and use another address to tell the program that the ISR is over and that it should return to the place it was prior to the interrupt and continue with normal execution. Different interfaces such SPI, CAN and UARTs had to be learnt along how timers work.

The LPC2119 was not the only new electronic component with which the author had to familiarize himself. ICs like the CAN differential converter and voltage translators had to be learned.

As a final result the PCB and all the hardware are fully operational.

4.2 Software

4.2.1 Low Level

4.2.1.1 GPS

With the use of a computer, the GPS was programmed to have a baud rate of 38400BPS. This is also the same baud rate used by UART0 so the same function prototype could be used for both UARTs. This baud rate is saved on its flash memory and a battery is used to keep the last track of the satellites in order to have a fast restart. At every startup the GPS is reconfigured to send only the GPGGA and the GPTGV strings. This is not necessary because this information is also saved on the flash memory but it was added as an extra security measurement. With these two strings and basic mathematics it is possible to acquire all three axis velocities and a heading value, in case the Magnetometer does not work. A UART interrupt reads the data from the GPS every second and functions activated by flags treat the data. The device should be initialized on an open area so it can have a satellite lock in less

time. After the signal is locked it can move to denser areas since it has a sensitivity of -146dBm for acquisition but increases to -158dBm for tracking.

4.2.1.2 ADIS16350

The ADIS16350 was successfully programmed to send all three linear accelerations and all three angular velocities. A timer with interrupt every 0.1 seconds samples the data at this same frequency. Functions, activated by the timer interrupt, transform the values from the sensor in actual accelerations and velocities. At every startup a 30 seconds calibration of the ADIS16350 will take place. For these 30 seconds the device should not have any type of movement. These 30 seconds is also used to give some extra time for the GPS to acquire a signal.

4.2.1.3 HMR3300

As mentioned before the magnetometer was not working properly at the moment of writing this paper. Nonetheless its initialization software and data acquisition was completed. Every time it is initialized it is configured to send data at its maximum baud rate, 19200BPS. After data acquisition/transfer the HMR3300 is placed into idle mode where it sends data only when requested which will be once per second. When GPS interrupt occurs it will activate a flag that will ask for data from HMR3300. Figure 11 can be used to follow the options from the acquisition of heading angle. The HMR3300 will send data through the UART1 which will activate a receiving interrupt. Another flag will tell the filter to use the HMR3300 heading data. If the flag is not activated, the sensor is broken or missing, so that the program will use the GPS heading data. After that the code continues normally into calculate the orthogonal velocities Vel-X and Vel-Y.

Figure 11 - HMR3300 Block Diagram

4.2.2 High Level

The high level consisted of translating the simulation model of the plane from MatLab into a C code program. Small parts of the MatLab code were translated to smaller functions in C using an IDE program. Like this it was possible to test multiple small parts of the code independently. After this these small function were optimized by various methods described previously and again tested individually. The maximum rounding errors of each step on a 6500 steps simulation can be seen on Figure 9. After testing they were put together into bigger functions and tested independently one more time. If the functions were working they were put together into even bigger functions until only the five original MatLab files could be represented by their cloned C functions. This procedure was very time consuming but assured that each part of the code would work independently and also together with other functions. This was very important because of the use of many global variables common to different functions.

4.2.3 Combining Low Level with High Level

To the moment of the writing of this paper the interface between the high level and low level software was not completely tested on the board. This was because of incompatibilities between the Dev-C++ and the Linux gcc. The Linux gcc does not accept some simple commands that were accepted by the Dev-C++ compiler. For example the setting up of interrupts is completely different on the Linux gcc from the others IDE windows based compilers. Besides this there is also the problem described before with the saturation of the RAM memory and all the long process needed for programming the flash memory of the LPC2119. The combination of Low and High level functions is thought to be trivial since the interface requires simply passing ten sensor outputs from the data acquisition step to the ExKalman function.

Using emulators and non-real data the programs runs continuous. The last problem found on the project is definitely the worst one. The incompatibilities mentioned above are do not only to the saturation of the RAM when the program is written in the RAM for testing. Also when the program is written to the flash it needs to store its variables into the RAM so it can use it through the program. The LPC2119 has 16Kb of RAM. This means that the total size of variables at one single time during process cannot be bigger than 16Kb. Only from the matrices used by the model of the airplane the total needed space is greater than 38Kb. This is because of the multiple temporary matrices needed of different sizes. Also the options of using GPS signal or no GPS signal also increases the number of matrices. An estimative of the number of matrices just in the high level part of the program is something in the order of 15 to 20 matrices of 22x22 floating variables. This is the same as 20*22*22*4 (one float equals 4 bytes) which equals approximately 39Kb. Making a wrought estimate that the total size of RAM needed to run the program is in the order of 60Kb. Not forgetting that the RAM is not only used for storing the global variables but also of new incoming and changing variables.

5 DIFFICULTIES

A dedicated chapter to explain the difficulties that were found during the development of the project that did not receive detailed attention on previous chapters.

5.1 Difficulties

5.1.1 Hardware

The problems in developing the hardware were mainly on learning how to deal with sensors and a microcontroller that were never used before by the author. The first problem was time – only three months were available for the project. Besides the problem of time the project faced the normal problems of a project that is starting from scratch. Many different types of sensors must be analyzed before a decision can be made to buy one. Sometimes it was even needed to change the sensor because a better one was found or because the one selected was not available. Every time the sensor was changed it was necessary not only to restart writing the code again but also to change the board schematics.

The complications in learning a new microcontroller family from scratch should not be underestimated. It took approximately 1 month to fully understand the LPC2119 microprocessor. After that it took even more time to understand the different protocols from the different communication processes available from the microcontroller. This includes SPI, CAN, UART, INTERRUPTS and TIMERS interfaces and protocols.

Studying different voltage levels from different sensors and correctly connecting them. In the board there are four different voltage levels. From the input that can vary from 7V to around 30V, to 5V for the GPS, 3.3V for the LPC2119 peripherals and

1.8V for the LPC2119 core. To identify which sensors use TTL or where information is sent at 3.3V but received at 5V took a lot of time.

Integrating everything described above into one full fully functional board using Eagle was another challenge. Just to learn how to use Eagle took around one month. Innumerous different types of boards were developed and redeveloped. Each one of them had some sort of incompatibility with other components. Small mistakes ranged from too small clearance space between one line and the ground to a too big distance between the capacitor and the sensor. Also learning how to make new devices in EAGLE was quite a challenge.

5.1.2 Software

From the software section most of the problems were already discussed on the above sections. They include but are not restricted to:

- Different compilers' incompatibility
- Lack of one main integrated IDE program that could program both with RAM or with FLASH using just one Operating System.
- Learning GPS protocols like NMEA and PMTK with incomplete datasheets.
- Studying the sensors and selecting appropriate filters and calibration methods.

5.1.3 Third Party

Most of the problems that occurred were due to the lack of time. Most of this lack of time happened because of small things that could not be controlled by the author. After the sensors were selected and they actually arrived, it took an extra 10 days to actually receive a connector so the sensor could be connected to a evaluation board. Some very important details about the programming of the GPS were not found on the datasheets from the manufactures' website. A complete datasheet was only provided by the manufacture on the 22nd of April. The PCB was received from production only on the 24th of April. By the 26th of April the board was completely assembled and tested. But it left only 27 days until the deadline for this paper.

6 CONCLUSION

A brief description of what was accomplished and what was not accomplished

From the hardware point of view the project is a success. The board is ready and fully tested. All the hardware works perfectly and all the sensors except the magnetometer are properly installed and configured. From the original proposal the only thing that has changed is that the magnetometer will connect to the main board through the UART0 and not through the SPI interface. But as mentioned in the beginning of the project this possibility was reserved as an option. Unfortunately the magnetometer is not properly calibrated so for the moment the heading is being provided by the GPS.

The low level software provides ten different measurements. They are:

- Vel-X
- Vel-Y
- Vel-Z
- Acc-X
- Acc-Y
- Acc-Z
- Gyro-X
- Gyro-Y
- Gyro-Z
- Heading.

This part is also ready and tested. The high level is ready and has been tested with pseudo data. To the present moment both parts of the program have been blended together but not yet fully tested. With the completion of the last part the values acquired from the low level will be applied to the model of the plane and from the model not only the angular position will be available, as a final result. The following measurements will be available to the UAV if needed:

- Vxc - velocity along X axis
- Vyc - velocity along Y axis
- Vzc - velocity along Z axis
- Ox - angular velocity along X axis
- Oy - angular velocity along Y axis
- Oz - angular velocity along Z axis
- theta - angle along Z axis
- gama - angle along X axis
- psi - angle along Y axis
- xg - distance to target
- h – attitude
- zg - side distance from course line
- Ogx - Ox gyro drift
- Ogy - Oy gyro drift
- Ogz - Oz gyro drift
- Ax - X accelerometer drift
- Ay - Y accelerometer drift
- Az - Z accelerometer drift
- Vax - integration of X accelerometer drift = Vx drift
- Vay - integration of Y accelerometer drift = Vy drift
- Vaz - integration of Z accelerometer drift = Vz drift
- uu - wind shaping filter state

Through emulation programs and fake data strings it was possible to test that the program works. Unfortunately the project the way it is now cannot be feasible. Do to the memory limitation and/or complexity of the plane model, the full program cannot be implemented. On the next session there are possible future correction and fixes to make the project feasible.

5 FUTURE CHALLENGES

Ideas for possible future implementations with the purpose of improving of the project.

An idea for a good Bachelors graduation thesis would be to study the results of the Navigational Unit and propose better error matrixes.

Blend the Navigational Unit to the Observation Camera in the UAV and program the interaction between them. As the plane moves the camera should correct its position.

Acquire a better GPS that uses pseudorange code and program the Navigational Unit with it. The pseudorange code has the capability of predicting where the future lock position will be, this would increase accuracy and also allow faster speeds for the plane.

Create a flag that tells the Navigational Unit that the GPS data is corrupted. This would be necessary because the GPS keeps sending data even if it has no satellites locked. It will send a string with zero values.

The easiest and cheapest possibility to fix the project would be to use a much simpler model of the plane. One possibility would be to use only the basic equations of movement as the model. The second approach would be to substitute.

Possible substitutes for the LPC2119:

LPC2106 – This processor has 60Kb of built in RAM but it does not has a CAN interface. Using a multiplex it would be possible to use one of the UARTs as a communication port. It has only one SPI so the Magnetometer needs to be on a UART port. It also has the same maximum frequency of 60 MHz.

LPC2210FBD144/01 – Mostly differs (focusing on the projects goals) from the LPC2106 because it has 2 SPI. Also has 64Kb of RAM.

LPC2220FBD144 – Only differs from the LPC2210FBD144/01 because it has a slightly higher clock frequency of up to 75MHz.

LPC2468FBD208 – One of the most powerful ARM7 type from NXP. Goes to 72MHz, has up to 70Kb for RAM, total of 96Kb of RAM. Also has 2xCAN, 1xI2S, 4xUART, 3xI2C, 3xSPI. It also has an SD/MMC interface which could be used for storing the data during flight for later analyses.

Another alternative is to jump to an ARM9 type processor. Most of the ARM9 processors from NXP have 60Kb or more of RAM, besides they all run at superior speed, which would improve accuracy and sampling rate.

A good option would be the TMS320F28235 or TMS320F28235 from Texas Instruments. It has 68Kb of RAM, runs at 150MHz. It has 1 SPI, 2 CAN, 3 UARTS and 512Kb of flash. It also has the option of 256Kb of flash.

7 BIBLIOGRAPHY

ADIS16350, A. D. ADIS16350 - Tri axis Inertial Sensor. Analog Devices.

Bijker, J. (2006). *A Low-Cost Integrated GPS/INS Navigation System for the Land Vehicle.* A Low-Cost Integrated GPS/INS Navigation System for the Land Vehicle.

CADSoft. (s.d.). *CADSoft Online*. Fonte: http://www.cadsoft.de/.

Cai, J. W. *A Low-Cost Integrated GPS/INS Navigation System for the Land Vehicle.* Beijing, China: School of Electronics and Information Engineering.

Chang-sun Yoo, l.-k. A. *LOW COST GPS/INS SENSOR FUSION SYSTEM FOR UAV NAVIGATION.* Daejon, Korea: Korea Aerospace Research Institute.

Europe, N. (2006). *MTK NMEA Packet User Manual.*

Europe, N. *NL-504ETTL User Manual.*

Gade, K. *Introduction To Inertial Navigation.* Forsvarets Forksningsinstitutt.

Gerais, F. U. (s.d.). *http://www.4shared.com/file/40698941/d7f7eb11/Apostila_-_Curso_Linguagem_C_-_UFMG.html*. Fonte: File:Apostila_-_Curso_Linguagem_C_-_UFMG.

HMR3300, H. HMR3300 Tri axis Magnetometer.

JÚNIOR, M. F. (2006). *IMPLEMENTAÇÃO DE CENTRAL INERCIAL.* Brasilia: Universidade de Brasilia.

Mayhew, D. M. *Multi-rate Sensor Fusion for GPS Navigation.* Virginia: Virginia Polytechnic Institute.

Mohinder S. Grewal, L. R. (January 2007). *Global Positioning Systems, Inertial Navigation and Integration.* Wiley.

Moore, J. B. *Direct Kalman Filtering Approach for GPS INS Integration.* Canberra, Australia: The Australian National University.

NXP, P. LPC21xx User Manual.

Scarlett, J. *Enhancing the Performance of Pedometers Using Single Accelerometer.* Norwood, MA: Analog Devices.

Smith, A. S. *Microelectrônica.* Makron Books.

Suh, Y. S. *Attitude Estimation Using Low Cost Accelerometer and Gyroscope.* Ulsan, Korea: Scholl of Electrical Enginnering.

Timex. (s.d.). *http://www.timex.ca/en/html/watch_inst_comp_DAI.html*. Fonte: http://www.timex.ca/en.

8 APPENDICES

APPENDIX 8. 1 - Declination Angles

(Timex)

	DECLINATION ANGLES	
Location	**Capital**	**Declination**
Afghanistan	Kabul	2-E
Albania	Tirana	3-E
Algeria	Algiers	1-E
Andorra	Andorra la Vella	1-W
Angola	Luanda	6-W
Antigua and Barbuda	Saint John's	14-W
Argentina	Buenos Aires	6-W
Armenia	Yerevan	5-E
Australia	Canberra	12-E
Austria	Vienna	2-E
Azerbaijan	Baku	5-E
Bahamas, The	Nassau	7-W
Bahrain	Manama	2-E
Bangladesh	Dhaka	0
Barbados	Bridgetown	15-W
Belarus	Minsk	6-E
Belgium	Brussels	1-W
Belize	Belmopan	1-E
Benin	Porto-Novo	4-W
Bhutan	Thimphu	0-E
Bolivia	La Paz (administrative)	4-W
Bolivia	Sucre (legislative/judiciary)	6-W
Bosnia and Herzegovina	Sarajevo	2-E
Botswana	Gaborone	15-W
Brazil	Brasília	19-W
Brunei	Bandar Seri Begawan	0
Bulgaria	Sofia	3-E
Burkina Faso	Ouagadougou	4-W
Burma	Rangoon	0.5-E
Burundi	Bujumbura	0
Cambodia	Phnom Penh	1-E
Cameroon	Yaoundé	3-W
Canada	Ottawa	14-W
Cape Verde	Praia	12-W
Central African Republic	Bangui	1-E
Chad	N'Djamena	1-E
Chile	Santiago	5-E
China	Beijing	6-W
Colombia	Bogotá	4-W
Comoros	Moroni	6-W
Congo (Brazzaville)	Brazzaville	1-W
Congo (Kinshasa)	Kinshasa	3-W
Costa Rica	San José	0
Côte d'Ivoire	Yamoussoukro	7-W
Croatia	Zagreb	2-E

Cuba	Havana	3-W
Cyprus	Nicosia	3-E
Czech Republic	Prague	2-E
Denmark	Copenhagen	1-E
Djibouti	Djibouti	1-E
Dominica	Roseau	14-W
Dominican Republic	Santo Domingo	10-W
Ecuador	Quito	1-E
Egypt	Cairo	3-E
El Salvador	San Salvador	3-E
Equatorial Guinea	Malabo	3-W
Eritrea	Asmara	2-E
Estonia	Tallinn	6-E
Ethiopia	Addis Ababa	1-E
Fiji	Suva	13-E
Finland	Helsinki	6-E
France	Paris	1-W
Gabon	Libreville	4-W
Gambia, The	Banjul	9-W
Georgia	T'bilisi	5-E
Germany	Berlin	1-E
Ghana	Accra	5-W
Greece	Athens	3-E
Grenada	Saint George's	14-W
Guatemala	Guatemala	3-E
Guinea	Conakry	9-W
Guinea-Bissau	Bissau	9-W
Guyana	Georgetown	15-W
Haiti	Port-au-Prince	8-W
Holy See	Vatican City	1-E
Honduras	Tegucigalpa	1-E
Hong Kong	Hong Kong	2-W
Hungary	Budapest	4-E
Iceland	Reykjavík	19-W
India	New Delhi	1-E
Indonesia	Jakarta	1-E
Iran	Tehran	3-E
Iraq	Baghdad	3.5-E
Ireland	Dublin	6-W
Israel	Jerusalem	3-E
Italy	Rome	1-E
Jamaica	Kingston	6-W
Japan	Tokyo	7-W
Jordan	Amman	3-E
Kazakhstan	Astana	9-E
Kenya	Nairobi	1-E
Kiribati	Tarawa	10-E
Korea, North	P'yongyang	8-W
Korea, South	Seoul	7-W
Kuwait	Kuwait	2-E
Kyrgyzstan	Bishkek	5-E
Laos	Vientiane	1-E
Latvia	Riga	5-E
Lebanon	Beirut	3-E
Lesotho	Maseru	21-W
Liberia	Monrovia	8-W
Libya	Tripoli	1-E
Liechtenstein	Vaduz	0-E
Lithuania	Vilnius	5-E
Luxembourg	Luxembourg	0

Macedonia, The Former Yugoslav Rep. of	Skopje	3-E
Madagascar	Antananarivo	15-W
Malawi	Lilongwe	5-W
Malaysia	Kuala Lumpur	1-E
Maldives	Male	3-W
Mali	Bamako	6-W
Malta	Valletta	1-E
Marshall Islands	Majuro	8-E
Mauritania	Nouakchott	8-W
Mauritius	Port Louis	19-W
Mexico	Mexico City	6-E
Micronesia, Federated States of	Palikir	6-E
Moldova	Chisinau	5-E
Monaco	Monaco	0
Mongolia	Ulaanbaatar	3-W
Montenegro	Podgorica	3-E
Morocco	Rabat	3-W
Mozambique	Maputo	18-W
Namibia	Windhoek	14-W
Nauru	Yaren District (no capital city)	9-E
Nepal	Kathmandu	0
Netherlands	Amsterdam	1-W
New Zealand	Wellington	22-E
Nicaragua	Managua	1-E
Niger	Niamey	3-W
Nigeria	Abuja	2-W
Norway	Oslo	0
Oman	Muscat	1-E
Pakistan	Islamabad	2-E
Palau	Koror	1-E
Panama	Panama	2-W
Papua New Guinea	Port Moresby	6-E
Paraguay	Asunción	11-W
Peru	Lima	1-E
Philippines	Manila	1-W
Poland	Warsaw	4-E
Portugal	Lisbon	5-W
Qatar	Doha	1-E
Romania	Bucharest	4-E
Russia	Moscow	9-E
Rwanda	Kigali	0
Saint Kitts and Nevis	Basseterre	14-W
Saint Lucia	Castries	14-W
Saint Vincent and the Grenadines	Kingstown	14-W
Samoa	Apia	11-E
San Marino	San Marino	1-E
Sao Tome and Principe	São Tomé	5-W
Saudi Arabia	Riyadh	2-E
Senegal	Dakar	0
Serbia	Belgrade	3-E
Seychelles	Victoria	5-W
Sierra Leone	Freetown	9-W
Singapore	Singapore	0
Slovakia	Bratislava	2-E
Slovenia	Ljubljana	2-E
Solomon Islands	Honiara	10-E
Somalia	Mogadishu	1-E
South Africa	Bloemfontein (judiciary)	21-W
South Africa	Cape Town (legislative)	23-W
South Africa	Pretoria (administrative)	17-W

Spain	Madrid	3-W
Sri Lanka	Colombo	3-W
Sudan	Khartoum	2-E
Suriname	Paramaribo	17-W
Swaziland	Lobamba (legislative)	19-W
Swaziland	Mbabane (administrative)	18-W
Sweden	Stockholm	3-E
Switzerland	Bern	0
Syria	Damascus	3-E
Taiwan	T'ai-pei	3-W
Tajikistan	Dushanbe	4-E
Tanzania	Dar es Salaam	2-W
Tanzania	Dodoma (legislative)	2-W
Thailand	Bangkok	0
Togo	Lomé	5-W
Tonga	Nuku'alofa	13-E
Trinidad and Tobago	Port-of-Spain	14-W
Tunisia	Tunis	1-E
Turkey	Ankara	4-E
Turkmenistan	Ashgabat	4-E
Tuvalu	Funafuti	11-E
Uganda	Kampala	0-E
Ukraine	Kiev	6-E
United Arab Emirates	Abu Dhabi	1-E
United Kingdom	London	3-W
United States	Washington, DC	10-W
Uruguay	Montevideo	8-W
Uzbekistan	Tashkent	5-E
Vanuatu	Port-Vila	12-E
Venezuela	Caracas	10-W
Vietnam	Hanoi	1-E
Yemen	Sanaa	1-E
Zambia	Lusaka	6-W
Zimbabwe	Harare	8-W

APPENDIX 8. 2 - NMEA Protocol

(Europe, NL-504ETTL User Manual)

Table 5.1-3 Position Fix Indicators

Value	Description
0	Fix not available or invalid
1	GPS SPS Mode, fix valid
2	Differential GPS, SPS Mode, fix valid
3-5	Not supported
6	Dead Reckoning Mode, fix valid

- **GLL--- Geographic Position – Latitude/Longitude**

Table 5.1-4 contains the values for the following example:

$GPGLL,2503.6319,N,12136.0099,E,053740.000,A,A*52

Table 5.1-4 GLL Data Format

Name	Example	Units	Description
Message ID	$GPGLL		GLL protocol header
Latitude	2503.6319		ddmm.mmmm
N/S indicator	N		N=north or S=south
Longitude	12136.0099		dddmm.mmmm
E/W indicator	E		E=east or W=west
UTC Time	053740.000		hhmmss.sss
Status	A		A=data valid or V=data not valid
Mode	A		A=autonomous, D=DGPS, E=DR
Checksum	*52		
<CR> <LF>			End of message termination

- **GSA---GNSS DOP and Active Satellites**

Table 5.1-5 contains the values for the following example:

$GPGSA,A,3,24,07,17,11,28,08,20,04,,,,,2.0,1.1,1.7*35

Table 5.1-5 GSA Data Format

Name	Example	Units	Description
Message ID	$GPGSA		GSA protocol header
Mode 1	A		See Table 5.1-6
Mode 2	3		See Table 5.1-7
ID of satellite used	24		Sv on Channel 1
ID of satellite used	07		Sv on Channel 2
....			....
ID of satellite used			Sv on Channel 12
PDOP	2.0		Position Dilution of Precision
HDOP	1.1		Horizontal Dilution of Precision
VDOP	1.7		Vertical Dilution of Precision
Checksum	*35		
<CR> <LF>			End of message termination

Table 5.1-6 Mode 1

Value	Description
M	Manual- forced to operate in 2D or 3D mode
A	Automatic-allowed to automatically switch 2D/3D

Table 5.1-7 Mode 2

Value	Description
1	Fix not available
2	2D
3	3D

- **GSV---GNSS Satellites in View**

Table 5.1-8 contains the values for the following example:

```
$GPGSV,3,1,12,28,81,285,42,24,67,302,46,31,54,354,,,20,51,077,46*73
$GPGSV,3,2,12,17,41,328,45,07,32,315,45,04,31,250,40,11,25,046,41*75
$GPGSV,3,3,12,08,22,214,38,27,08,190,16,19,05,092,33,23,04,127,*7B
```

Table 5.1-8 GSV Data Format

Name	Example	Units	Description
Message ID	$GPGSV		GSV protocol header
Total number of messages[1]	3		Range 1 to 3
Message number[1]	1		Range 1 to 3
Satellites in view	12		
Satellite ID	28		Channel 1 (Range 01 to 32)
Elevation	81	degrees	Channel 1 (Range 00 to 90)
Azimuth	285	degrees	Channel 1 (Range 000 to 359)
SNR (C/No)	42	dB-Hz	Channel 1 (Range 00 to 99, null when not tracking)
Satellite ID	20		Channel 4 (Range 01 to 32)
Elevation	51	degrees	Channel 4 (Range 00 to 90)
Azimuth	077	degrees	Channel 4 (Range 000 to 359)
SNR (C/No)	46	dB-Hz	Channel 4 (Range 00 to 99, null when not tracking)
Checksum	*73		
<CR> <LF>			End of message termination

1. Depending on the number of satellites tracked multiple messages of GSV data may be required.

- **RMC---Recommended Minimum Specific GNSS Data**

Table 5.1-9 contains the values for the following example:

```
$GPRMC,053740.000,A,2503.6319,N,12136.0099,E,2.69,79.65,100106,,,A*53
```

Table 5.1-9 RMC Data Format

Name	Example	Units	Description
Message ID	$GPRMC		RMC protocol header
UTC Time	053740.000		hhmmss.sss
Status	A		A=data valid or V=data not valid
Latitude	2503.6319		ddmm.mmmm
N/S Indicator	N		N=north or S=south
Longitude	12136.0099		dddmm.mmmm
E/W Indicator	E		E=east or W=west
Speed over ground	2.69	knots	True
Course over ground	79.65	degrees	
Date	100106		ddmmyy
Magnetic variation		degrees	
Variation sense			E=east or W=west (Not shown)
Mode	A		A=autonomous, D=DGPS, E=DR
Checksum	*53		
<CR> <LF>			End of message termination

- **VTG---Course Over Ground and Ground Speed**

Table 5.1-10 contains the values for the following example:

$GPVTG,79.65,T,,M,2.69,N,5.0,K,A*38

Table 5.1-10 VTG Data Format

Name	Example	Units	Description
Message ID	$GPVTG		VTG protocol header
Course over ground	79.65	degrees	Measured heading
Reference	T		True
Course over ground		degrees	Measured heading
Reference	M		Magnetic
Speed over ground	2.69	knots	Measured speed
Units	N		Knots
Speed over ground	5.0	km/hr	Measured speed
Units	K		Kilometer per hour
Mode	A		A=autonomous, D=DGPS, E=DR
Checksum	*38		
<CR> <LF>			End of message termination

APPENDIX 8. 3 - MTK Protocol

(Europe, MTK NMEA Packet User Manual, 2006)

Pkt Type	Abbreviation / Syntax	Data Field	Meaning / Example / Return
101	PMTK_CMD_HOT_START	None	Hot Restart: Use all available data in the NV storage ***$PMTK101*32<CR><LF>***
102	PMTK_CMD_WARM_START	None	Warm Restart: Don't use Ephemeris at re-start ***$PMTK102*31<CR><LF>***
103	PMTK_CMD_COLD_START	None	Cold Restart: Don't use Time, Position, Almanacs and Ephemeris data at re-start ***$PMTK103*30<CR><LF>***
104	PMTK_CMD_FULL_COLD_START	None	Essentially a Cold Restart, but additionally clear system/user configurations at re-start. Reset the receiver to factory default *$PMTK104*37<CR><LF>*
251	PMTK_SET_NMEA_BAUDRATE ***PMTK251,Baudrate***	Baudrate: Baud rate setting 0 – default setting, 4800, 9600, 14400, 19200, 38400, 57600, 115200	Set NMEA port baud rate ***$PMTK251,38400*27<CR><LF>***
300	PMTK_API_SET_FIX_CTL ***PMTK300,FixInterval,0,0,0,0***	FixInterval: Position fix interval [msec]. Must be larger than 200	This parameter controls the rate of position fixing activity ***$PMTK300,1000,0,0,0,0*1C<CR><LF>***
301	PMTK_API_SET_DGPS_MODE ***PMTK301,Mode***	Mode: DGPS data source mode. 0 : No DGPS source 1 : RTCM 2 : WAAS	DGPS correction data source mode ***$PMTK301,1*2D<CR><LF>***
313	PMTK_API_SET_SBAS_ENABLED	0 = Disable 1 = Enable	Enable to search a SBAS satellite or not ***$PMTK313,1*2E<CR><LF>***
314	PMTK_API_SET_NMEA_OUTPUT	See below note 1	Set NMEA sentence output frequency ***$PMTK314,1,1,1,1,1,5,1,1,1,1,1,1,0,1,1,1,1,1*2C<CR><LF>***
320	PMTK_API_SET_PWR_SAV_MODE ***PMTK320,Mode*** (Internal testing ONLY)	Mode: 0: power saving mode off 1: power saving mode on	Set power saving operation mode ***$PMTK320,0*26<CR><LF>***
330	PMTK_API_SET_DATUM ***PMTK330,Datum***	Datum: 0 : WGS84 1 : TOKYO-M 2 : TOKYO-A	Set default datum ***$PMTK330,0*2E<CR><LF>***
331	PMTK_API_SET_DATUM_ADVANCE ***PMTK331,majA,ecc,dX,dY,dZ***	majA: User defined datum semi-major axis [m] ecc: User defined datumeccentric [m] dX: User defined datum to WGS84 X axis offset [m] dY: User defined datum to WGS84 X axis offset [m] dZ: User defined datum to WGS84 X axis offset [m]	Set user defined datum ***$PMTK331, 6377397.155, 299.1528128, -148.0, 507.0,685.0*16<CR><LF>***

390	PMTK_API_SET_USER_OPTION ***PMTK390, Lock, Update_Rate, Baud_Rate, GLL_Period, RMC_Period, VTG_Period, GSA_Period, GSV_Period, GGA_Period, ZDA_Period, MCHN_Period***	Lock: nonzero: freeze the setting; 0: allow further setting Update_Rate: 1~5 (Hz) Baud_Rate: 115200, 57600, 38400, 19200, 14400, 9600, 4800 XXX_Period: NMEA sentence output period	Write the user setting to the flash to override the default setting. Maximum 8 times without erase the chip. This feature may not be available ***$PMTK390,1,1,38400,1,1,1,1,1,1,1,0*2B<CR><LF>***

Note 1:

Total 19 data fields representing output frequency for each of the 19 supported NMEA sentences.

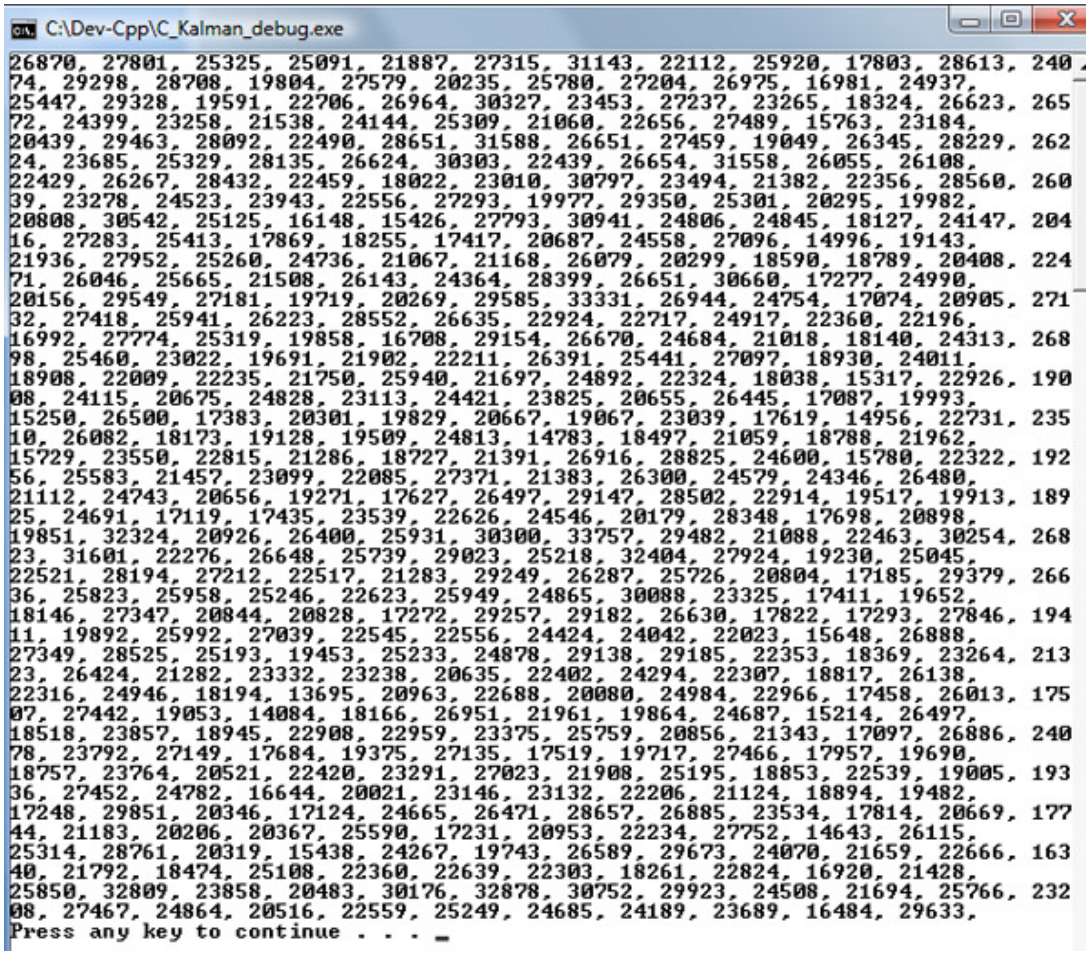
0 NMEA_SEN_GLL, // GPGLL interval - Geographic Position - Latitude longitude
1 NMEA_SEN_RMC, // GPRMC interval - Recommended Min. specific GNSS sentence
2 NMEA_SEN_VTG, // GPVTG interval - Course Over Ground and Ground Speed
3 NMEA_SEN_GGA, // GPGGA interval - GPS Fix Data
4 NMEA_SEN_GSA, // GPGSA interval - GNSS DOPS and Active Satellites
5 NMEA_SEN_GSV, // GPGSV interval - GNSS Satellites in View
6 NMEA_SEN_GRS, // GPGRS interval - GNSS Range Residuals
7 NMEA_SEN_GST, // GPGST interval - GNSS Pseudorange Error Statistics
13 NMEA_SEN_MALM, // PMTKALM interval - GPS almanac information
14 NMEA_SEN_MEPH, // PMTKEPH interval - GPS ephemeris information
15 NMEA_SEN_MDGP, // PMTKDGP interval - GPS differential correction information
16 NMEA_SEN_MDBG, // PMTKDBG interval – MTK debug information
17 NMEA_SEN_ZDA, // GPZDA interval – Time & Date
18 NMEA_SEN_MCHN, // PMTKCHN interval – GPS channel status

APPENDIX 8. 4 - nelinvyvojP Function example MatLab

Row\Colum	1	2	3	4	5	6	7	8	9	10	11
1	2.687	2.7801	2.5325	2.5091	2.1887	2.7315	3.1143	2.2112	2.592	1.7803	2.8613
2	2.5447	2.9328	1.9591	2.2706	2.6964	3.0327	2.3453	2.7237	2.3265	1.8324	2.6623
3	2.0439	2.9463	2.8092	2.249	2.8651	3.1588	2.6651	2.7459	1.9049	2.6345	2.8229
4	2.2429	2.6267	2.8432	2.2459	1.8022	2.301	3.0797	2.3494	2.1382	2.2356	2.856
5	2.0808	3.0542	2.5125	1.6148	1.5426	2.7793	3.0941	2.4806	2.4845	1.8127	2.4147
6	2.1936	2.7952	2.526	2.4736	2.1067	2.1168	2.6079	2.0299	1.859	1.8789	2.0408
7	2.0156	2.9549	2.7181	1.9719	2.0269	2.9585	3.3331	2.6944	2.4754	1.7074	2.0905
8	1.6992	2.7774	2.5319	1.9858	1.6708	2.9154	2.667	2.4684	2.1018	1.814	2.4313
9	1.8908	2.2009	2.2235	2.175	2.594	2.1697	2.4892	2.2324	1.8038	1.5317	2.2926
10	1.525	2.65	1.7383	2.0301	1.9829	2.0667	1.9067	2.3039	1.7619	1.4956	2.2731
11	1.5729	2.355	2.2815	2.1286	1.8727	2.1391	2.6916	2.8825	2.46	1.578	2.2322
12	2.1112	2.4743	2.0656	1.9271	1.7627	2.6497	2.9147	2.8502	2.2914	1.9517	1.9913
13	1.9851	3.2324	2.0926	2.64	2.5931	3.03	3.3757	2.9482	2.1088	2.2463	3.0254
14	2.2521	2.8194	2.7212	2.2517	2.1283	2.9249	2.6287	2.5726	2.0804	1.7185	2.9379
15	1.8146	2.7347	2.0844	2.0828	1.7272	2.9257	2.9182	2.663	1.7822	1.7293	2.7846
16	2.7349	2.8525	2.5193	1.9453	2.5233	2.4878	2.9138	2.9185	2.2353	1.8369	2.3264
17	2.2316	2.4946	1.8194	1.3695	2.0963	2.2688	2.008	2.4984	2.2966	1.7458	2.6013
18	1.8518	2.3857	1.8945	2.2908	2.2959	2.3375	2.5759	2.0856	2.1343	1.7097	2.6886
19	1.8757	2.3764	2.0521	2.242	2.3291	2.7023	2.1908	2.5195	1.8853	2.2539	1.9005
20	1.7248	2.9851	2.0346	1.7124	2.4665	2.6471	2.8657	2.6885	2.3534	1.7814	2.0669
21	2.5314	2.8761	2.0319	1.5438	2.4267	1.9743	2.6589	2.9673	2.407	2.1659	2.2666
22	2.585	3.2809	2.3858	2.0483	3.0176	3.2878	3.0752	2.9923	2.4508	2.1694	2.5766
Row\Colum	12	13	14	15	16	17	18	19	20	21	22
1	2.4074	2.9298	2.8708	1.9804	2.7579	2.0235	2.578	2.7204	2.6975	1.6981	2.4937
2	2.6572	2.4399	2.3258	2.1538	2.4144	2.5309	2.106	2.2656	2.7489	1.5763	2.3184

3	2.6224	2.3685	2.5329	2.8135	2.6624	3.0303	2.2439	2.6654	3.1558	2.6055	2.6108
4	2.6039	2.3278	2.4523	2.3943	2.2556	2.7293	1.9977	2.935	2.5301	2.0295	1.9982
5	2.0416	2.7283	2.5413	1.7869	1.8255	1.7417	2.0687	2.4558	2.7096	1.4996	1.9143
6	2.2471	2.6046	2.5665	2.1508	2.6143	2.4364	2.8399	2.6651	3.066	1.7277	2.499
7	2.7132	2.7418	2.5941	2.6223	2.8552	2.6635	2.2924	2.2717	2.4917	2.236	2.2196
8	2.6898	2.546	2.3022	1.9691	2.1902	2.2211	2.6391	2.5441	2.7097	1.893	2.4011
9	1.9008	2.4115	2.0675	2.4828	2.3113	2.4421	2.3825	2.0655	2.6445	1.7087	1.9993
10	2.351	2.6082	1.8173	1.9128	1.9509	2.4813	1.4783	1.8497	2.1059	1.8788	2.1962
11	1.9256	2.5583	2.1457	2.3099	2.2085	2.7371	2.1383	2.63	2.4579	2.4346	2.6481
12	1.8925	2.4691	1.7119	1.7435	2.3539	2.2626	2.4546	2.0179	2.8348	1.7698	2.0898
13	2.6823	3.1601	2.2276	2.6648	2.5739	2.9023	2.5218	3.2404	2.7924	1.923	2.5045
14	2.6636	2.5823	2.5958	2.5246	2.2623	2.5949	2.4865	3.0088	2.3325	1.7411	1.9652
15	1.9411	1.9892	2.5992	2.7039	2.2545	2.2556	2.4424	2.4042	2.2023	1.5648	2.6888
16	2.1323	2.6424	2.1282	2.3332	2.3238	2.0635	2.2402	2.4294	2.2307	1.8817	2.6138
17	1.7507	2.7442	1.9053	1.4084	1.8166	2.6951	2.1961	1.9864	2.4687	1.5214	2.6497
18	2.4078	2.3792	2.7149	1.7684	1.9375	2.7135	1.7519	1.9717	2.7466	1.7957	1.969
19	1.9336	2.7452	2.4782	1.6644	2.0021	2.3146	2.3132	2.2206	2.1124	1.8894	1.9482
20	1.7744	2.1183	2.0206	2.0367	2.559	1.7231	2.0953	2.2234	2.7752	1.4643	2.6115
21	1.634	2.1792	1.8474	2.5108	2.236	2.2639	2.2303	1.8261	2.2824	1.692	2.1428
22	2.3208	2.7467	2.4864	2.0516	2.2559	2.5249	2.4685	2.4189	2.3689	1.6484	2.9633

APPENDIX 8. 5 - nelinvyvojP Function example in C (10^{-4})



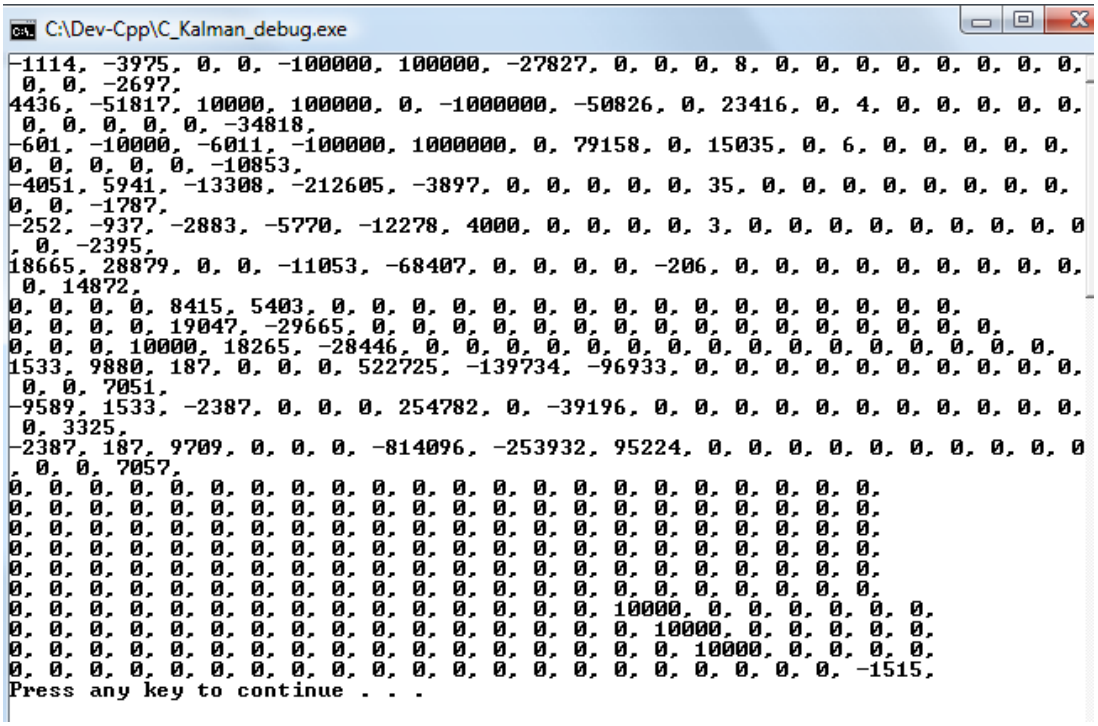
APPENDIX 8. 6 - Substituce1 Function example MatLab

Row\Colum	1	2	3	4	5	6	7	8	9	10	11
1	-0.1114	-0.3975	0	0	-10	10	-2.78	0	0	0	0.0008
2	0.4436	-5.1817	1	10	0	-100	-5.08	0	2.34	0	0.0004
3	-0.0601	-1	-0.6011	-10	100	0	7.916	0	1.5	0	0.0006
4	-0.4051	0.5941	-1.3308	-21.2605	-0.3897	0	0	0	0	0	0.0035
5	-0.0252	-0.0937	-0.2883	-0.577	-1.2278	0.4	0	0	0	0	0.0003
6	1.8665	2.8879	0	0	-1.1053	-6.8407	0	0	0	0	-0.0206
7	0	0	0	0	0.8415	0.5403	0	0	0	0	0
8	0	0	0	0	1.9047	-2.9665	0	0	0	0	0
9	0	0	0	1	1.8265	-2.8446	0	0	0	0	0
10	0.1533	0.988	0.0187	0	0	0	52.27	-14	-9.7	0	0
11	-0.9589	0.1533	-0.2387	0	0	0	25.48	0	-3.9	0	0
12	-0.2387	0.0187	0.9709	0	0	0	-81.4	-25.4	9.52	0	0
13	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0

Row\Colum	12	13	14	15	16	17	18	19	20	21	22
1	0	0	0	0	0	0	0	0	0	0	-0.2697
2	0	0	0	0	0	0	0	0	0	0	-3.4818
3	0	0	0	0	0	0	0	0	0	0	-1.109
4	0	0	0	0	0	0	0	0	0	0	-0.1787
5	0	0	0	0	0	0	0	0	0	0	-0.2395
6	0	0	0	0	0	0	0	0	0	0	1.4872
7	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0.7051
11	0	0	0	0	0	0	0	0	0	0	0.3325
12	0	0	0	0	0	0	0	0	0	0	0.7057
13	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0	1	0	0	0	0	0	0
20	0	0	0	0	0	1	0	0	0	0	0

21	0	0	0	0	0	0	1	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	-0.1515

APPENDIX 8. 7 - Substituce1 Function example in C (10^{-4})


C:\Dev-Cpp\C_Kalman_debug.exe
-1114, -3975, 0, 0, -100000, 100000, -27827, 0, 0, 0, 8, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, -2697,
4436, -51817, 10000, 100000, 0, -1000000, -50826, 0, 23416, 0, 4, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, -34818,
-601, -10000, -6011, -100000, 1000000, 0, 79158, 0, 15035, 0, 6, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, -10853,
-4051, 5941, -13308, -212605, -3897, 0, 0, 0, 0, 0, 35, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, -1787,
-252, -937, -2883, -5770, -12278, 4000, 0, 0, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, -2395,
18665, 28879, 0, 0, -11053, -68407, 0, 0, 0, 0, -206, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 14872,
0, 0, 0, 0, 8415, 5403, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 19047, -29665, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 10000, 18265, -28446, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
1533, 9880, 187, 0, 0, 0, 522725, -139734, -96933, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 7051,
-9589, 1533, -2387, 0, 0, 0, 254782, 0, -39196, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 3325,
-2387, 187, 9709, 0, 0, 0, -814096, -253932, 95224, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
, 0, 0, 7057,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

APPENDIX 8. 9 - ModelLetA Function example in C (10^{-4})